

Windows Kernel Fuzzing for Beginners

Ben Nagy

About Me:

- Not oldsk00l. Just old.
- ~ 5 weeks experience with Windows Kernel
- > 5 years experience with Fuzzing
- Hate all Technology
- Ruby and Drinking Make the Pain Go Away

Disclaimer:

I am aware of the prevailing opinion that fuzzing talks without bugs suck, by definition. I do not have any bugs. Even if I did have bugs, I wouldn't tell you. There are no bugs. There are, however, otters and buff Russian men of dubious sexuality. Also, many red boxes. You have been warned.



Secret Fuzzing Wisdoms

- Select a Good Target
- Acquire Essential Knowledge
- Apply Fuzzing Canon
 - How do we Deliver
 - How do we Instrument
 - How do we Generate
 - How does that Scale

Secret Fuzzing Wisdoms

- Delivery, Instrumentation, Generation
 - Gotta keep em separated!
 - Please stop writing heavily coupled tools, kthx
- A good toolchain allows rapid retargeting
 - Start fuzzing with a stupid generator
 - Cold cores find no bugs!

Target Selection

$$n_bugs = p_bug * n_tests$$

- p_bug / testing speed is inherently target specific
- Can tune the equation
 - Better (possibly slower) Generators
 - More Scale
 - Rapid Tooling (lead time counts!)
 - Better Samples
 - Pre Fuzzing Toolchain

p_bug++

- Feedback Driven Fuzzing
 - Via code coverage, success rate or some other metric
 - Eg SAGE, bunny, EFS, Flayer
 - PRO - Awesome, super elite, finds bugs dumb fuzzers will never hit
 - CON - Slow and difficult to write, poor windows support
- Fault Injection / deeply instrumented fuzzing
 - Inject bad data close to code being attacked
 - PRO - vastly simplifies delivery
 - CON - need to then check reachability
- Corpus Distillation
 - Low effort, high reward technique
 - Need a way to measure coverage (tricky for kernel stuff)

Target Selection

$$n_bugs = p_bug * n_tests$$

- More broadly, n_bugs isn't interesting
- Are there USEFUL bugs in there?
- If there are, can we locate them
 - Bug Chaff
 - Post Fuzzing Toolchain

Target Selection

$$n_bugs = p_bug * n_tests$$

- Bug Utility is SUBJECTIVE
- Sell? Use? Fix? Disclose?
- Whatever our utility metric, can we REALISE VALUE
 - Will it provide USEFUL CAPABILITY?
 - Is it RELIABLY exploitable?
 - Will anyone buy it anyway?
 - Is it worth fixing?
 - Will it bring us fame and imply great sexual prowess?

Windows Kernel, Simplified

- Featuring “Barry the Kernel Otter”
- Some stuff is completely missing or wrong
- All of it is greatly simplified
- Real resources abound!
 - MSDN (new layout / navigation is awesome)
 - Anything by j00ru, Alex Ionescu, Tarjei Mandt
 - Anything by Russinovich / Solomon / Probert
 - “CRK” is an academic course, freely downloadable
 - “WRK” is a full windows kernel source tree, plus build tools

Userland

kernel32

ntdll

“NT Executive”

Dragons

Hardware

Userland

kernel32

ntdll

1. Setup syscall args
2. syscall number in eax
3. int2e / sysenter /
syscall

("context switch")

"NT Executive"

4. Lookup syscall in SSDT
5. Dispatch to correct
driver

Dragons

Hardware

Userland

kernel32

ntdll

"NT Executive"

IO

USER

GDI

Dragons

Other Complicated Stuff

Hardware

Userland

kernel32

ntdll

"NT Executive"

IO

USER

GDI

Dragons

Drivers

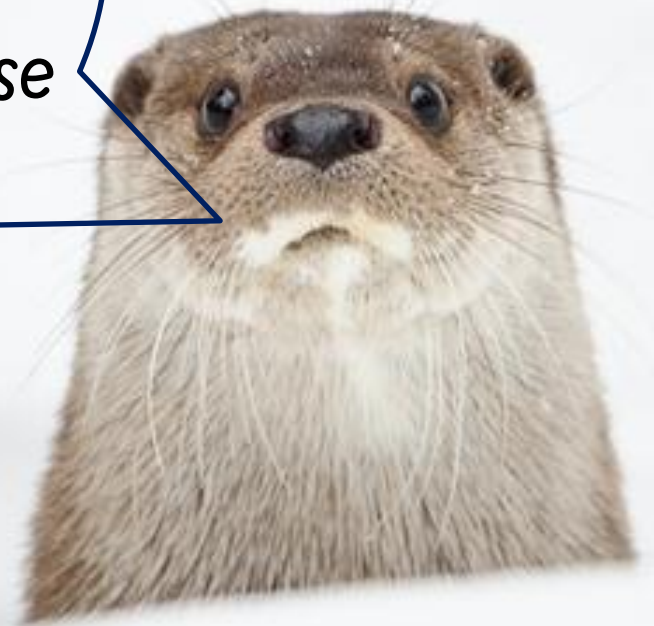
Are

Layered!

Other Complicated Stuff

Hardware

- Windows IO is deeply async
- Uses IO Request Packets (IRP)
- “Filter” Drivers can intercept these



Userland

user32

"NT Executive"

IO

USER

GDI

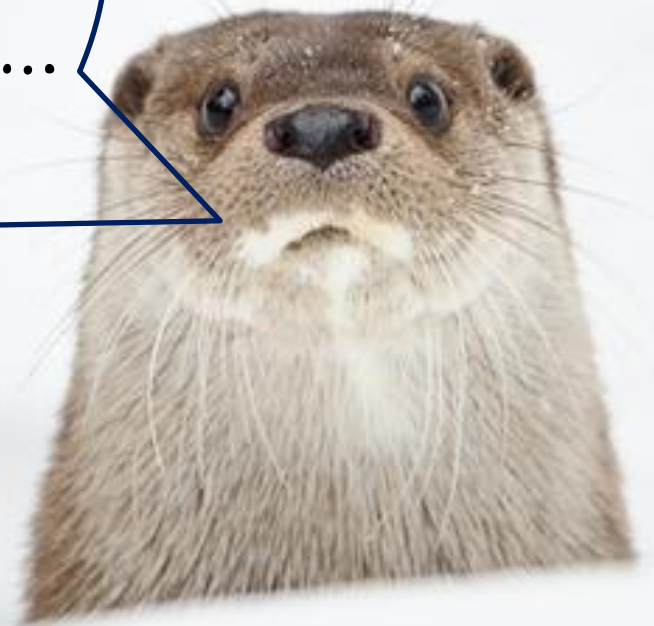
Daddy Issues

Repressed Memories

Hardware

USER runs the GUI

- Windows, Menus, Cursors, Icons...



Userland

gdi32

"NT Executive"

IO

USER

GDI

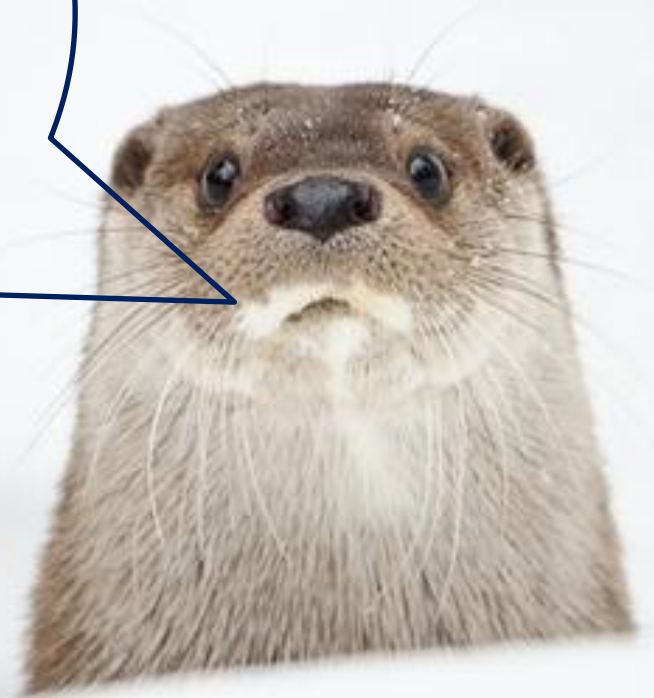
Unladen Swallows

Meaning of Life

Hardware

Graphics Driver Interface

- Basically, it draws stuff
- Moved into kernel space ~NT4
- Bitmaps, Fonts, Metafiles...



Userland

user32 / gdi32

"NT Executive"

IO

USER

GDI

Broccoli

Drivers

Win32k.sys

Are

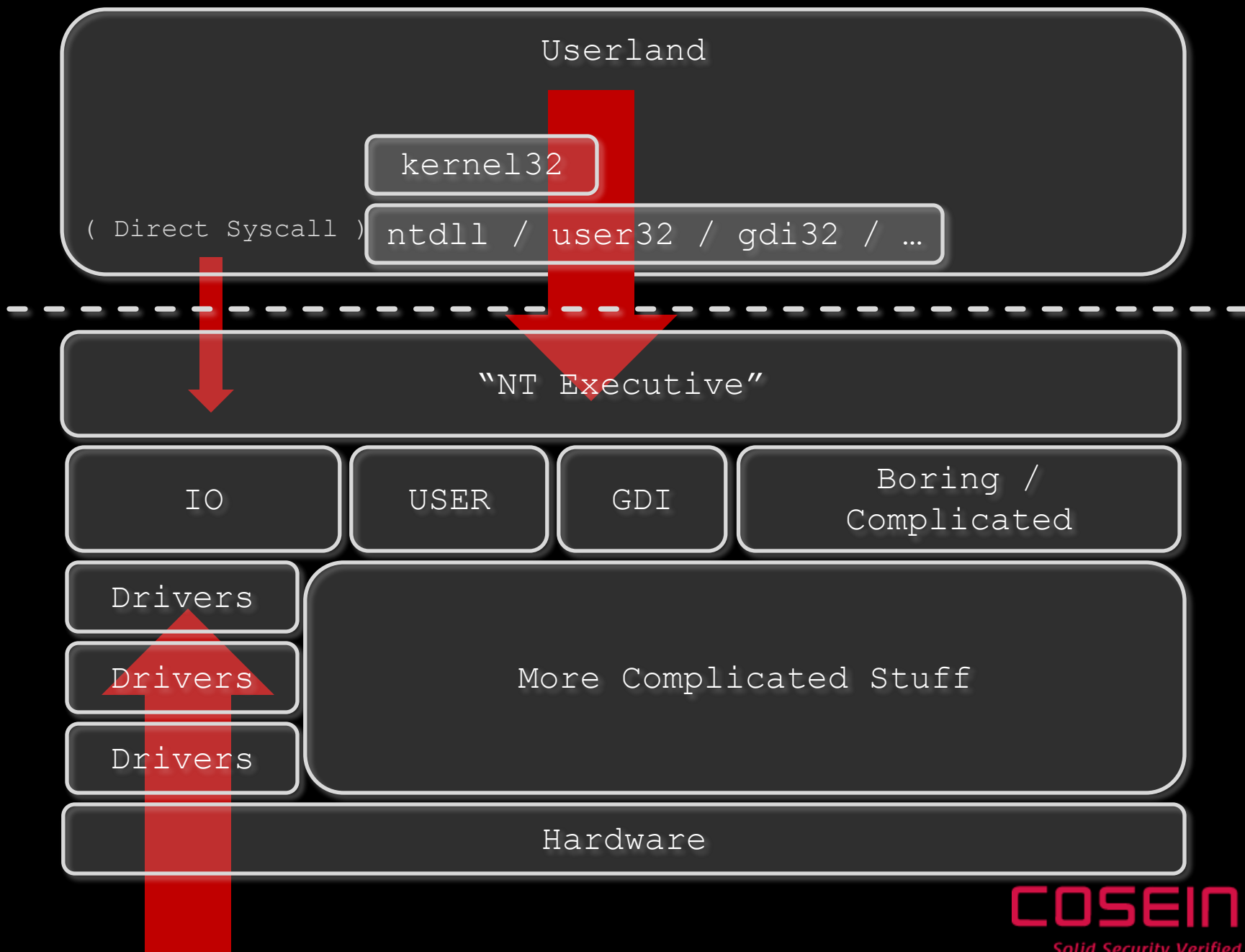
Evil Clowns

Layered!

Hardware

A man and a woman are dancing in a disco setting. The woman is wearing a black short-sleeved top and a dark, pleated skirt. The man is wearing a white long-sleeved shirt with a dark collar and white trousers. They are both smiling and looking towards the camera. The background is a wooden wall with a large red light source on the left. The floor is a light-colored, polished surface.

Disco



Userland

kernel32

ntdll / user32 / gdi32 / ...

Hook?

"NT Executive"

Hook?

IO

USER

GDI

Boring /
Complicated

Drivers
Filter?

Drivers

Drivers

More Complicated Stuff

Hardware

Bug Classes

- LocalLocal
 - Privilege escalation
 - Sandbox escapes
 - Trending upwards in importance
- RemoteRemote
 - Used to be the shiznit, now plagued by issues
 - Firewalls
 - Were great for indiscriminate attacks, less for targeted
- RemoteLocal
 - Require a user to do something
 - Attack via email, document, URL etc
 - Now the Rolls Royce of bugs

Attack Vector Evaluation

- Coming 'up' from the hardware side
 - Will yield RemoteRemotes
 - Just like 'normal' network fuzzing
 - SMB, RDP, tcpip.sys, wifi, USB...
 - Reliability issues? Stealth?
 - Hardware differences?

Verdict: You first, guv.

Attack Vector Evaluation

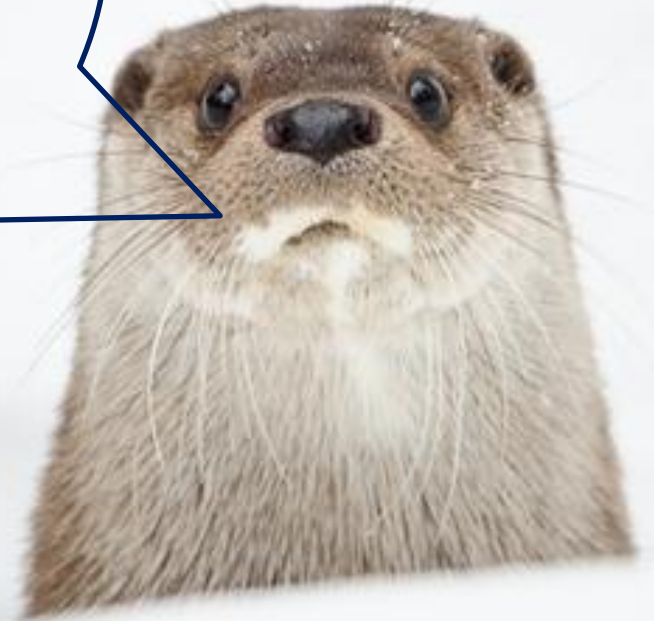
- SSDT Hooks / Filter Drivers / etc
 - Good for attacking 3rd party drivers
 - Fuzzing logic itself really should be in-kernel (inflexible)
 - Public implementations available
 - <http://code.google.com/p/ioctlfuzzer>
- Finding AV bugs seems too cruel to be sport
- Can't write drivers in Ruby ☹️

Attack Vector Evaluation

- GDI is cool, because RemoteLocals
 - Historically bug prone
- General Syscalls might be fun
 - LocalLocals, but easy to prototype
- USER is tricky, only yields LocalLocals
 - Keyboard Layouts burned by Stuxnet
 - Plus, Tarjei already looked at it

(Moment of Silence in honour of Bug Genocide)

Let's hit GDI!!



GDI - Delivery Vectors

- Here's what I have so far
 - Fonts - TTF, OTF, FON....
 - Cursors - BMP, CUR (animated)
 - Metafiles - EMF, WMF
 - Images - JPEG, PNG (!!)
- Not even close to complete

GDI - Fonts

- Great slides from BHEU12

http://media.blackhat.com/bh-eu-12/Lee/bh-eu-12-Lee-GDI_Font_Fuzzing-Slides.pdf

(MANY THANKS to Lee & Chan for also sharing code)

- Fonts are tricky beasts
- You can also embed them (google EOT)
- Simple 9 step process...

GDI - Fonts

1. Load the fuzzed font from a file

```
debug_info "Removing any old copies of  
GDI.RemoveFontResourceEx(font_file, 0, nil) #  
added=GDI.AddFontResourceEx(font_file, 0, nil)
```

- I'm NOT using FR_PRIVATE
- Works for almost any font type
- Protip - fix checksums
 - (google B1B0AFBA)

GDI - Window Basics

2. Create a Window Callback

```
def window_proc(hwnd, umsg, wparam, lparam)
  case umsg
    when GDI::WM_DESTROY
      GDI.PostQuitMessage(0)
      return 0
    else
      # This handles all messages we don't explicitly
      process
      return GDI.DefWindowProc(hwnd, umsg, wparam,
        lparam)
    end
  end
end
```

GDI - Window Basics

- Lots of people put their logic in here
 - Handle WM_PAINT, WM_RESIZE etc
 - Lots of samples online do it this way, too...
- I never found the need, but YMMV

GDI - Window Basics

3. Register Window Class

```
window_class = GDI::WNDCLASSEX.new  
window_class[:lpfnWndProc] =  
window_class[:hInstance] = hinst  
window_class[:hbrBackground] = GDI::COLOR_WINDOW  
window_class[:hCursor] = 0
```

```
@atom = GDI.RegisterClassEx( window_class )
```

GDI - Window Basics

4. Create a Window Instance

```
@hwnd ||= GDI.CreateWindowEx(  
    GDI::WS_EX_LEFT,                                # extended style  
    poi(@atom),                                     # class name or  
    atom                                             #  
    @opts[:title],                                  # window title  
    GDI::WS_OVERLAPPEDWINDOW | GDI::WS_VISIBLE,    # style  
    GDI::CW_USEDEFAULT,                             # X pos  
    GDI::CW_USEDEFAULT,                             # Y pos  
    @opts[:width],                                  # width  
    @opts[:height],                                 # height  
    0,                                                # parent  
    0,                                                # menu  
    hinst,                                           # instance  
    nil                                              # lparam  
)
```

GDI - Fonts

5. Get Font Face Name (undocumented)

```
success=GDI.GetFontResourceInfo(  
    w_fname,  
    sz,  
    buf,  
    2 # asks to receive a LOGFONTW in buf  
)  
lf=LOGFONTW.new buf # cast the buffer to a  
LOGFONTW  
GDI.WideCharToMultiByte( ...  
lf[:lfFaceName].to_ptr ...)
```

GDI - Fonts

6. "Create" the Font

```
logical_font = GDI::LOGFONTW.new
logical_font[:lfHeight] = font_size
logical_font[:lfFaceName].to_ptr.put_string(0, font_face)
logical_font[:lfItalic] = 0
logical_font[:lfCharSet] = GDI::DEFAULT_CHARSET
```

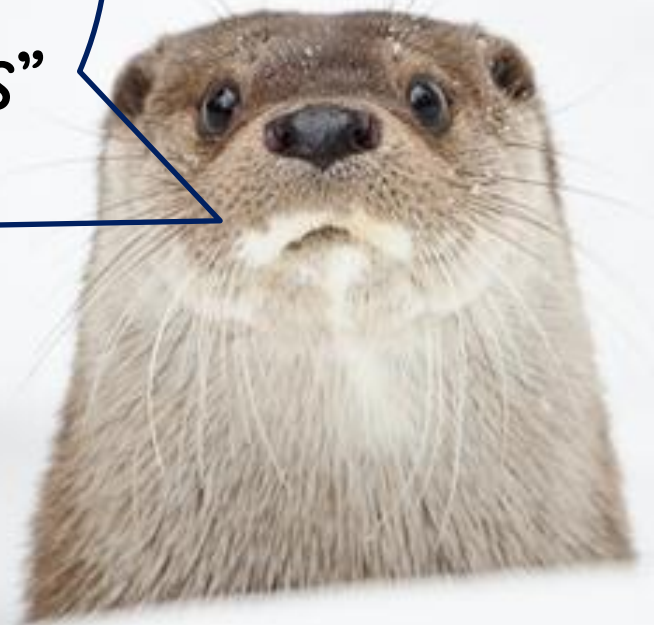
```
@current_font=GDI.CreateFontIndirect logical_font
raise_win32_error if @current_font.zero?
```

7. Select it into the DC for our window

```
@old_font=GDI.SelectObject(dc, @current_font)
```

What are Device Contexts?

- Bits of screen or printer
- Include “graphics attributes”
- (eg brushes, fonts, etc)



GDI - Fonts

8. How big is a 'line' of text?

```
# build the string one glyph at a time until the
# text extent is greater than our rect width
sz = GDI::SIZE.new
until sz[:cx] > width || str.empty?
  out << str.slice!( 0,1 )
  GDI.GetTextExtentPoint32( dc, out, out.size,
sz )
  guess = out.size
end
```

GDI - Fonts

9. Actually draw some f**king text

```
GDI.send(  
    text_out_method,          # ExtTextOutW / A  
    dc,                      # device context  
    0,                       # X start  
    @current_y,              # Y start  
    GDI::ETO_GLYPH_INDEX,    # For 'raw' mode  
    this_line,               # RECT  
    out,                     # str to draw  
    out.size,                # size  
    nil                      # lpDx  
)  
@current_y+=sz[:cy]
```

ETO_GLYPH_INDEX

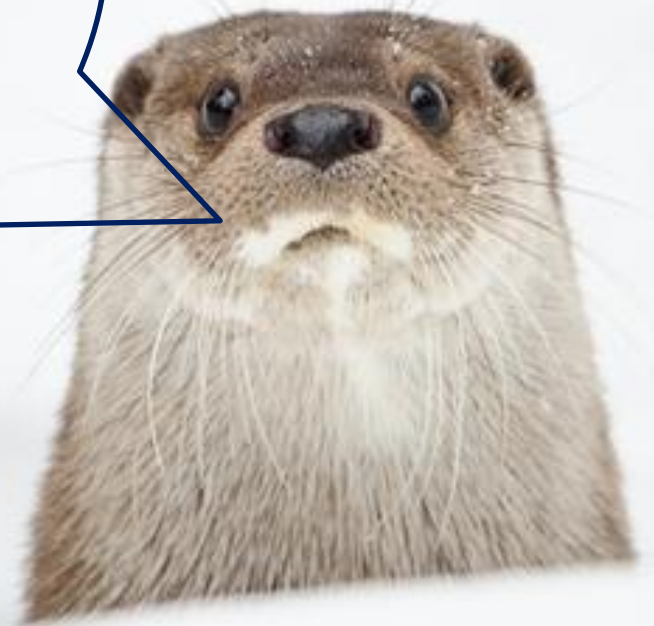
“ The lpString array refers to an array returned from GetCharacterPlacement and should be parsed directly by GDI as no further language-specific processing is required. ”

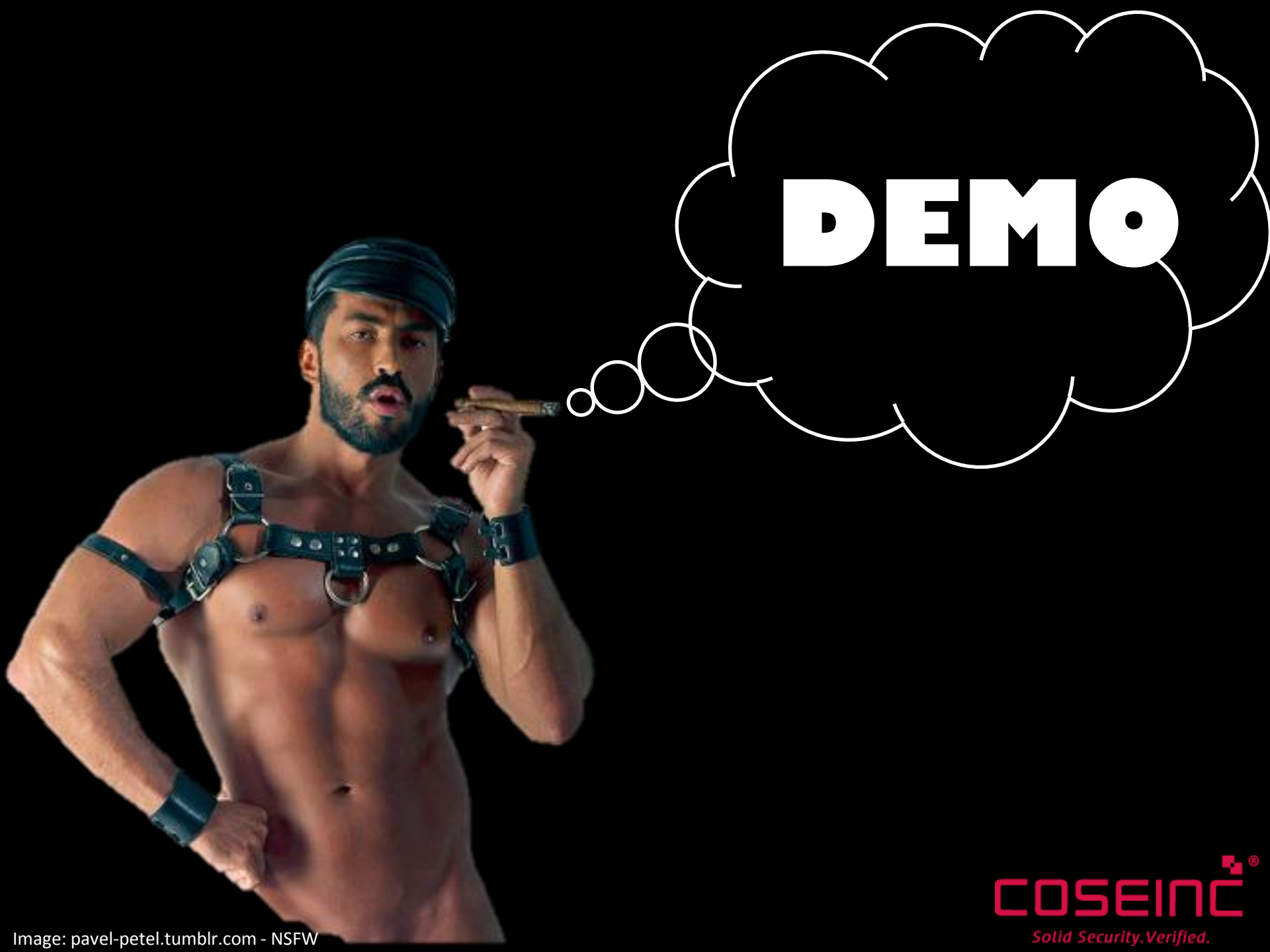
— MSDN

(This is why we use ExtTextOut and not DrawText)

That Sucked!

(Still better than Gtk tho)



A muscular man with a beard and a black headband is shown from the waist up. He is wearing a black leather harness with multiple straps and buckles across his chest and shoulders. He is holding a cigar in his right hand, which is raised towards his face. A thought bubble originates from the cigar, containing the word "DEMO" in large, bold, white capital letters. The background is solid black.

DEMO

GDI - Cursors

```
hCursor=GDI.LoadCursorFromFile  
raise_win32_error if hCursor.zero?  
@old_cursor=GDI.SetCursor hCursor  
debug_info "Set cursor #{cursor_file}"
```

- WTF? Why no DC?
 - The cursor is a shared resource!
 - Not supposed to change it unless mouse is over you
 - Pff, whatever.

GDI - Cursors

```
@old_clip = GDI::RECT.new
@clip      = GDI::RECT.new
GDI.SetForegroundWindow @hwnd # _try_ to get
GDI.GetClipCursor @old_clip
GDI.GetWindowRect @hwnd, @clip
GDI.ClipCursor @clip          # Clipping changes
GDI.ClipCursor @old_clip # Put it back
```

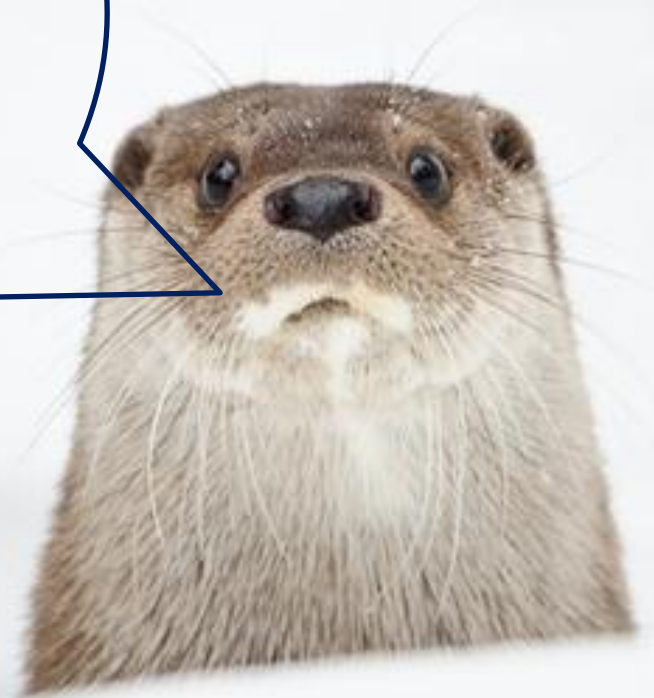
- Really crappy / fragile method!
 - Works, though



DEMO

Metafiles!

- Like a 'script' of GDI commands
- 'Scalable' == 'Fun'
- ~~Set~~AbortProc used to be lolz



GDI - Metafiles - WMF

```
if wmf_data[0..3] ==  
    debug_info "Aldus Placeable  
    pdata = pstr( wmf_data[22..-1] )
```

- WMF has no scaling / position data
- APM header is a standard 'nonstandard'
- Provides the missing info

Cannot the Scaling! What do?

1. Play in MSPAINT.EXE

- Uses GDI+ internally, converts to BMP
- Draws the BMP to the DC

2. Use Coordinate Spaces & Transforms API

- Parse the APM Header
- Do lots of annoying maths with pels and twips
- Actually, just saying 'pels' and 'twips' is annoying

3. Convert to EMF, play that

- May lose some evil, but very easy to do

GDI - Metafiles - WMF & EMF

```
emf_handle = GDI.SetWinMetaFileBits(  
    pdata.size,  
    pdata,  
    dc,  
    nil  
) # convert to EMF if required...  
raise_win32_error if emf_handle.zero?  
GDI.PlayEnhMetaFile dc, emf_handle, rect  
GDI.DeleteEnhMetaFile emf_handle
```



DEMO

GDI - JPEG / PNG

The **StretchDIBits** function copies the color data for a rectangle of pixels in a DIB, **JPEG, or PNG** image to the specified destination rectangle. If the destination rectangle is larger than the source rectangle, this function stretches the rows and columns of color data to fit the destination rectangle. If the destination rectangle is smaller than the source rectangle, this function compresses the rows and columns by using the specified **raster operation**.

- MSDN



GDI - JPEG / PNG

To ensure proper metafile spooling while **printing**, applications must call the `CHECKJPEGFORMAT` or `CHECKPNGFORMAT` escape to verify that the **printer** recognizes the JPEG or PNG image, respectively, before calling **StretchDIBits**.

- MSDN



Fine. Let's be a Printer.

1. (Optional) Get default printer

```
buf=pstr( "\x00" * 260 )
buf_sz=FFI::MemoryPointer.new( :ulong )
buf_sz.write_ulong buf.size
if GDI.GetDefaultPrinter buf, buf_sz
  buf.read_string buf=pstr( "\x00" * 260 )
...
```

(Or just specify "Fax" etc)

Fine. Let's be a Printer.

2. (Optional) Check for JPEG Support

```
escape_code=FFI::MemoryPointer.new :ulong
escape_code.write_ulong GDI::CHECKJPEGFORMAT
# Check if CHECKJPEGFORMAT exists
res=GDI.ExtEscape(
  printer_dc,
  GDI::QUERYESCSUPPORT,
  escape_code.size,
  escape_code,
  0,
  nil
)
if res > 0
  status=FFI::MemoryPointer.new :ulong
  res=GDI.ExtEscape(
    printer_dc,
    GDI::CHECKJPEGFORMAT,
    p_jpeg_data.size,
    p_jpeg_data,
    status.size,
    status
  )
```

Yes, I realise you can't read this....

Just use one of the built-in printers like XPS or OneNote, they support JPEG.

3. Fill Out Bitmap Info Struct

```
bmi_header = GDI::BITMAPINFOHEADER.new
bmi_header[:biSize] = GDI::BITMAPINFOHEADER.size
bmi_header[:biWidth] = img_width
# top down image - negative height value
bmi_header[:biHeight] = -img_height
bmi_header[:biPlanes] = 1
bmi_header[:biBitCount] = 0
bmi_header[:biCompression] = GDI::BI_JPEG
bmi_header[:biSizeImage] = img_data.bytesize
```

4. Do the Thing

```
printer_dc=GDI.CreateDC nil, lpszDevice, nil, nil  
retval=GDI.StretchDIBits(  
    printer_dc,  
    0, # dest X  
    0, # dest Y  
    stretch_width || rand(1000), # width  
    stretch_height || rand(1000), # height  
    0, # src X  
    0, # src Y  
    img_width,  
    img_height,  
    pstr( img_data ),  
    bmi_header,  
    GDI::DIB_RGB_COLORS, GDI::SRCCOPY  
)
```

If this returns > 0 then it is “scan lines copied”, which should be the same as your JPEG height. Yay.



NO DEMO

One More Thing...

```
# first 4 args are passed in registers.
register_args=args.shift( 4 ).zip %w( rcx rdx r8 r9 )
register_args.map! {|arg,reg| "mov #{reg}, #{arg}" }
# the rest are passed on the stack
stack_args=args.reverse.map {|arg| "push #{arg}"}
stub_x64=[
  "mov r10, rcx",           # don't know why
  "mov eax, #{syscall}",    # syscall in eax
  "syscall",                # make the call
  "add rsp, #{stack_args.size * 8}", # clean up the stack
  "ret"
]
```

```
asm = (register_args + stack_args + stub_x64).join "\n"
opcodes = Metasm::Shellcode.assemble(
  Metasm::X86_64.new, asm
).encode_string
p_opcodes = FFI::MemoryPointer.from_string opcodes
```

One More Thing...

```
Syscall.VirtualProtect(  
    p_opcodes,  
    p_opcodes.size,  
    PAGE_EXECUTE_READWRITE,  
    FFI::MemoryPointer.new( DWORD ) # receives old protection value  
)  
hThread = Syscall.CreateThread(  
    nil,  
    0,  
    p_opcodes,  
    nil,  
    CREATE_SUSPENDED,  
    nil  
)  
self.raise_win32_error if hThread.zero?  
Syscall.CloseHandle hThread
```

1 Line Syscall Fuzzer!

```
Syscall.call64  
  rand(0x2000),
```

```
* (Array.new(6).map {rand  
  2**32}) until @bsod
```

Basic technique stolen from jduck's MS10-073 exploit,
updated to work on x86 / x64. Also, props to Metasm team.

Out of time!!

- Did not talk about...
- Case Generation
 - I mainly use 'Millerfuzz' & Radamsa from OUSPUG
 - (and secret stuff)
- Scale
 - Scaling by VM pairs has proved fragile
 - I use 'checkpoints' with auto-reboot on BSOD
 - You can test with NotMyFault tool
 - Uncleared dump + checkpoints sent for analysis
 - VMs don't always reboot cleanly ☹
 - Private WER server may be better?

kthxbai

- As I mentioned, 5 weeks ago I knew ~nothing about the kernel
- Anything I got right is probably thanks to:
 - Lee & Chan for their code from BHEU12
 - Tarjei Mandt, Alex Ionescu, jduck
 - New MSDN Nagivation Interface
 - Luck



</talk>



(ben at coseinc dot com)

COSEINC
Solid Security.Verified.